IN THE CLAIMS:

Please rewrite the claims to read as follows:

1. (Currently Amended) A method for executing uniprocessor (UP) coded workloads in a multiprocessor (MP) computer system without having to rewrite the UP-coded workloads' code, comprising:

identifying threads in the uniprocessor coded workloads (UP-workloads) which can execute concurrently, and identifying threads in the UP-workloads which cannot execute concurrently;

assigning first threads which cannot execute concurrently to a first concurrency group, and assigning second threads which cannot execute concurrently to a second concurrency group, where any thread in the first concurrency group can execute concurrently with any thread in the second concurrency group;

~~organizing the UP-coded workloads into one or more concurrency groups, wherein UP-coded workloads in the same concurrency group are not permitted to execute concurrently with one another in the MP computer system;~~

compiling the UP workload to execute on the MP system, the threads in the first concurrency group compiled to be prevented from executing concurrently, the threads in the second concurrency group compiled to be prevented from executing concurrently, and the threads in the first and the second concurrency groups compiled to execute concurrently;

scheduling first and second execution vehicles that respectively execute on different processors in the MP computer system at substantially the same time;

acquiring the [[~~a~~]] first concurrency group by the first execution vehicle and the [[~~a~~]] second concurrency group by the second execution vehicle; and

executing UP-coded workloads in the first concurrency group through the first execution vehicle at substantially the same time as UP-coded workloads in the second concurrency group are executed through the second execution vehicle.

2. (Original) The method according to claim 1, wherein the UP-coded workloads are UP-coded threads, and the first and second execution vehicles are first and second processes.

3. (Original) The method according to claim 1, wherein the UP-coded workloads are messages, and the first and second execution vehicles are first and second threads.

4. (Original) The method according to claim 1, wherein the step of acquiring the first and second concurrency groups further comprises:

dequeueing from a concurrency-group run queue a first concurrency-group data structure associated with the first concurrency group; and

dequeueing from the concurrency-group run queue a second concurrency-group data structure associated with the second concurrency group.

5. (Original) The method according to claim 4, further comprising:

setting a first CG flag in the first concurrency-group data structure to a value indicating that the first concurrency group is in a running state; and

setting a second CG flag in the second concurrency-group data structure to a value indicating that the second concurrency group is in a running state.

6. (Original) The method according to claim 4, further comprising:

appending UP-coded workloads enqueued on a first current queue in the first concurrency-group data structure onto a first active queue in the first concurrency-group data structure; and

appending UP-coded workloads enqueued on a second current queue in the second concurrency-group data structure onto a second active queue in the second concurrency-group data structure.

7. (Original) The method according to claim 6, further comprising:

dequeueing UP-coded workloads in the first and second concurrency groups from the first and second active queues, respectively; and

executing the dequeued UP-coded workloads to completion.

8. (Original) The method according to claim 5, further comprising:

in response to the first execution vehicle finishing execution of the UP-coded workloads in the first concurrency group, the first execution vehicle performing the steps:

A) if at least one UP-coded workload in the first concurrency group is executable:

(i) setting the value of the first CG flag to a value indicating that the first concurrency group is in a queued state;

(ii) re-enqueueing the first concurrency-group data structure onto the concurrency-group run queue;

B) if there are not any UP-coded workloads in the first concurrency group that are executable, setting the first CG flag to a value indicating that the first concurrency group is in a suspended state;

C) dequeueing from the concurrency-group run queue a third concurrency-group data structure associated with a third concurrency group; and

D) setting a third CG flag in the third concurrency-group data structure to a value indicating that the third concurrency group is in a running state.

9. (Original) The method according to claim 1, wherein at least one of the UP-coded workloads is organized into the one or more concurrency groups at run-time.

10. (Original) The method according to claim 1, wherein the MP computer system is a network cache.

11. (Currently Amended) A multiprocessor (MP) computer system configured to execute a computer code derived from uniprocessor (UP) coded threads without having to rewrite the UP-coded threads' code, the MP computer system having the computer code, comprising:

a plurality of processors;

threads in the uniprocessor coded workloads (UP-workloads) which can execute concurrently, and threads in the UP-workloads which cannot execute concurrently;

first threads which cannot execute concurrently assigned to a first concurrency group, and second threads which cannot execute concurrently assigned to a second concurrency group, where any thread in the first concurrency group can execute concurrently with any thread in the second concurrency group;

the UP workload compiled to execute on the MP system, the threads in the first concurrency group compiled to be prevented from executing concurrently, the threads in the second concurrency group compiled to be prevented from executing concurrently, and the threads in the first and the second concurrency groups compiled to execute concurrently;

a memory having a plurality of storage locations addressable by the plurality of processors for storing data and program code, the memory being configured to store a separate concurrency-group data structure for each of the first and second ~~a plurality of~~ concurrency groups, each concurrency-group data structure ~~comprising~~: having, an active-queue pointer storing a location in the memory of an active queue of UP-coded thread messages associated with UP-coded threads in an executable state; and

a current-queue pointer storing a location in the memory of a current queue of UP-coded thread messages associated with UP-coded threads waiting to be transferred to the active queue.

12. (Original) The MP computer system according to claim 11, wherein each concurrency-group data structure further comprises a CG flag that stores a value indicating an operational state of a concurrency group associated with the concurrency-group data structure.

13. (Original) The MP computer system according to claim 11, wherein each UP-coded thread message stored in the active queue and current queue stores a location in the memory of a top of a call stack associated with a specific UP-coded thread.

14. (Original) The MP computer system according to claim 13, wherein the call stack is accessible through a thread control block (TCB) associated with the specific UP-coded thread, the TCB including a CG pointer for storing a memory location of a concurrency-group data structure.

15. (Original) The MP computer system according to claim 11, wherein each concurrency-group data structure further comprises meta-data information associated with a concurrency group.

16. (Original) The MP computer system according to claim 11, wherein the MP computer system is a network cache.

17. (Currently Amended) An apparatus for executing uniprocessor (UP) coded workloads in a multiprocessor (MP) computer system without having to rewrite the UP-coded workloads' code, comprising:

means for organizing the UP-coded workloads into one or more concurrency groups, wherein UP-coded workloads in the same concurrency group are not permitted to execute concurrently with one another in the MP computer system;

means for identifying threads in the uniprocessor coded workloads (UP-workloads) which can execute concurrently, and identifying threads in the UP-workloads which cannot execute concurrently;

means for assigning first threads which cannot execute concurrently to a first concurrency group, and assigning second threads which cannot execute concurrently to a second concurrency group, where any thread in the first concurrency group can execute concurrently with any thread in the second concurrency group;

~~means for organizing the UP-coded workloads into one or more concurrency groups, wherein UP-coded workloads in the same concurrency group are not permitted to execute concurrently with one another in the MP computer system;~~

means for compiling the UP workload to execute on the MP system, the threads in the first concurrency group compiled to be prevented from executing concurrently, the threads in the second concurrency group compiled to be prevented from executing concurrently, and the threads in the first and the second concurrency groups compiled to execute concurrently;

means for scheduling first and second execution vehicles that respectively execute on different processors in the MP computer system at substantially the same time;

means for acquiring the [[~~a~~]] first concurrency group by the first execution vehicle and the [[~~a~~]] second concurrency group by the second execution vehicle; and

means for executing UP-coded workloads in the first concurrency group through the first execution vehicle at substantially the same time as UP-coded workloads in the second concurrency group are executed through the second execution vehicle.

18. (Original) The apparatus according to claim 17, wherein the UP-coded workloads are UP-coded threads, and the first and second execution vehicles are first and second processes.

19. (Original) The apparatus according to claim 17, wherein the UP-coded workloads are messages, and the first and second execution vehicles are first and second threads.

20. (Original) The apparatus according to claim 17, further comprising:

means for dequeueing from a concurrency-group run queue a first concurrency-group data structure associated with the first concurrency group; and

means for dequeueing from the concurrency-group run queue a second concurrency-group data structure associated with the second concurrency group.

21. (Original) The apparatus according to claim 20, further comprising:

means for setting a first CG flag in the first concurrency-group data structure to a value indicating that the first concurrency group is in a running state; and

means for setting a second CG flag in the second concurrency-group data structure to a value indicating that the second concurrency group is in a running state.

22. (Original) The apparatus according to claim 20, further comprising:

means for appending UP-coded workloads enqueued on a first current queue in the first concurrency-group data structure onto a first active queue in the first concurrency-group data structure; and

means for appending UP-coded workloads enqueued on a second current queue in the second concurrency-group data structure onto a second active queue in the second concurrency-group data structure.

23. (Original) The apparatus according to claim 22, further comprising:

means for dequeueing UP-coded workloads in the first and second concurrency groups from the first and second active queues, respectively; and

means for executing the dequeued UP-coded workloads to completion.

24. (Original) The apparatus according to claim 21, further comprising:

means for setting the value of the first CG flag to a value indicating that the first concurrency group is in a queued state or in a suspended state; and

means for re-enqueueing the first concurrency-group data structure onto the concurrency-group run queue.

25. (Currently Amended) A computer-readable media comprising instructions for execution in one or more processors for executing uniprocessor (UP) coded workloads in a multiprocessor (MP) computer system without having to rewrite the UP-coded workloads' code, comprising:

identifying threads in the uniprocessor coded workloads (UP-workloads) which can execute concurrently, and identifying threads in the UP-workloads which cannot execute concurrently;

assigning first threads which cannot execute concurrently to a first concurrency group, and assigning second threads which cannot execute concurrently to a second concurrency group, where any thread in the first concurrency group can execute concurrently with any thread in the second concurrency group;

~~organizing the UP-coded workloads into one or more concurrency groups, wherein UP-coded workloads in the same concurrency group are not permitted to execute concurrently with one another in the MP computer system;~~

compiling the UP workload to execute on the MP system, the threads in the first concurrency group compiled to be prevented from executing concurrently, the threads in the second concurrency group compiled to be prevented from executing concurrently, and the threads in the first and the second concurrency groups compiled to execute concurrently;

scheduling first and second execution vehicles that respectively execute on different processors in the MP computer system at substantially the same time;

acquiring the [[~~a~~]] first concurrency group by the first execution vehicle and the [[~~a~~]] second concurrency group by the second execution vehicle; and

executing UP-coded workloads in the first concurrency group through the first execution vehicle at substantially the same time as UP-coded workloads in the second concurrency group are executed through the second execution vehicle.

26. (Original) The computer-readable media according to claim 25, wherein the UP-coded workloads are UP-coded threads, and the first and second execution vehicles are first and second processes.

27. (Original) The computer-readable media according to claim 25, wherein the UP-coded workloads are messages, and the first and second execution vehicles are first and second threads.

28. (Currently Amended) A method for executing workloads in a multiprocessor (MP) computer system, comprising:

identifying threads in an uniprocessor coded workloads (UP-workloads) which can execute concurrently, and identifying threads in the UP-workloads which cannot execute concurrently;

assigning first threads which cannot execute concurrently to a first concurrency group, and assigning second threads which cannot execute concurrently to a second concurrency group, where any thread in the first concurrency group can execute concurrently with any thread in the second concurrency group;

compiling the UP workload to execute on the MP system, the threads in the first concurrency group compiled to be prevented from executing concurrently, the threads in the second concurrency group compiled to be prevented from executing concurrently, and the threads in the first and the second concurrency groups compiled to execute concurrently;

~~organizing the workloads into one or more concurrency groups, wherein workloads in the same concurrency group are not permitted to execute concurrently with one another in the MP computer system;~~

scheduling first and second execution vehicles that respectively execute on different processors in the MP computer system at substantially the same time;

acquiring the [[a]] first concurrency group by the first execution vehicle and the [[~~a~~]] second concurrency group by the second execution vehicle; and executing workloads in the first concurrency group through the first execution vehicle at substantially the same time as workloads in the second concurrency group are executed through the second execution vehicle.

29. (Original) The method according to claim 28, wherein the step of acquiring the first and second concurrency groups further comprises:

dequeueing from a concurrency-group run queue a first concurrency-group data structure associated with the first concurrency group; and

dequeueing from the concurrency-group run queue a second concurrency-group data structure associated with the second concurrency group.

30. (Original) The method according to claim 29, further comprising:

setting a first CG flag in the first concurrency-group data structure to a value indicating that the first concurrency group is in a running state; and

setting a second CG flag in the second concurrency-group data structure to a value indicating that the second concurrency group is in a running state.

31. (Original) The method according to claim 29, further comprising:

appending workloads enqueued on a first current queue in the first concurrency-group data structure onto a first active queue in the first concurrency-group data structure; and

appending workloads enqueued on a second current queue in the second concurrency-group data structure onto a second active queue in the second concurrency-group data structure.

32. (Original) The method according to claim 31, further comprising:

dequeueing workloads in the first and second concurrency groups from the first and second active queues, respectively; and

executing the dequeued workloads to completion.

33. (Original) The method according to claim 30, further comprising:

in response to the first execution vehicle finishing execution of the workloads in the first concurrency group, the first execution vehicle performing the steps:

A) if at least one workload in the first concurrency group is executable:

(i) setting the value of the first CG flag to a value indicating that the first concurrency group is in a queued state;

(ii) re-enqueueing the first concurrency-group data structure onto the concurrency-group run queue;

B) if there are not any workloads in the first concurrency group that are executable, setting the first CG flag to a value indicating that the first concurrency group is in a suspended state;

C) dequeueing from the concurrency-group run queue a third concurrency-group data structure associated with a third concurrency group; and

D) setting a third CG flag in the third concurrency-group data structure to a value indicating that the third concurrency group is in a running state.

34. (Currently Amended) A method, comprising:

identifying threads in a uniprocessor coded workload (UP-workload) which can execute concurrently, and identifying threads in the UP-workload which cannot execute concurrently;

assigning first threads which cannot execute concurrently to a first concurrency group, and assigning second threads which cannot execute concurrently to a second concurrency group, where any thread in the first concurrency group can execute concurrently with any thread in the second concurrency group, and repeating assigning threads which cannot execute concurrently to further concurrency groups to form a plurality of concurrency groups, where any thread in a concurrency group can execute concurrently with any thread in a different concurrency group;

compiling the UP workload to execute on the MP system, the threads in the first concurrency group compiled to be prevented from executing concurrently, the threads in the second concurrency group compiled to be prevented from executing concurrently, and the threads in the first and the second concurrency groups compiled to execute concurrently, and repeating to prevent threads in any of the plurality of concurrency groups from executing concurrently, and permitting threads in different concurrency groups to execute concurrently;

~~organizing a plurality of workloads into a plurality of concurrency groups, wherein each workload in the same concurrency group are not permitted to execute concurrently with another workload in a microprocessor (MP) computer system;~~

scheduling a plurality of execution vehicles that respectively execute on different processors in the MP computer system at substantially the same time;

acquiring by each execution vehicle of the plurality of execution vehicles a concurrency group from the plurality of concurrency groups; and

executing workloads in the plurality of concurrency groups through the plurality of execution vehicles at substantially the same time.

35. (Previously Presented) The method according to claim 34, wherein the workloads are uniprocessor (UP) coded threads, and the plurality of vehicles are processes.

36. (Previously Presented) The method according to claim 34, wherein the workloads are messages, and the plurality of vehicles are first and second threads.

37. (Previously Presented) The method according to claim 34, wherein the workloads are uniprocessor (UP) coded workloads.